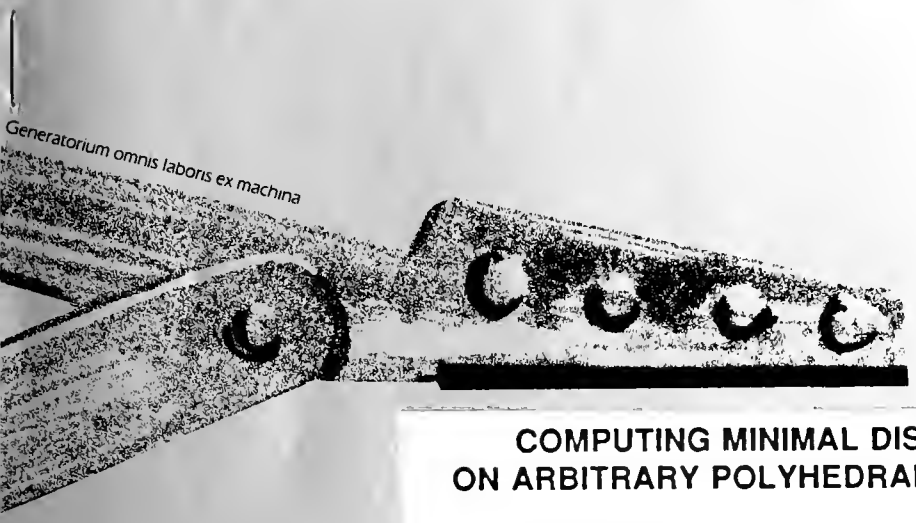
Robotics Research Technical Report



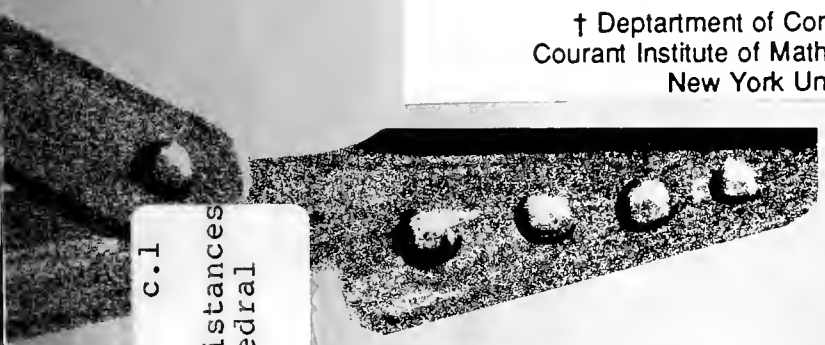
COMPUTING MINIMAL DISTANCES ON ARBITRARY POLYHEDRAL SURFACES

by

Estarose Wolfson ‡
Eric L. Schwartz † ‡

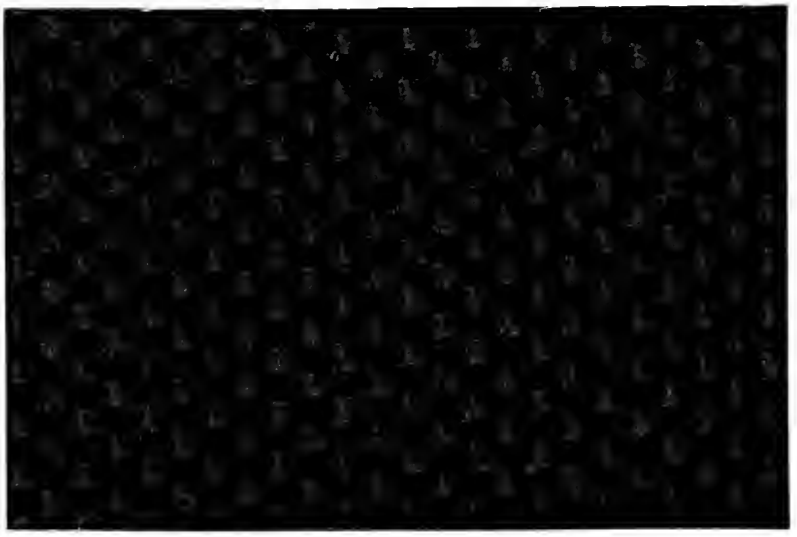
Technical Report No. 274
Robotics Report No. 96
January, 1987

† Deptartment of Computer Science
Courant Institute of Mathematical Sciences
New York University



New York University
Courant Institute of Mathematical Sciences

Computer Science Division
251 Mercer Street New York, N.Y. 10012



COMPUTING MINIMAL DISTANCES ON ARBITRARY POLYHEDRAL SURFACES

by

Estarose Wolfson ‡
Eric L. Schwartz † ‡

Technical Report No. 274
Robotics Report No. 96
January, 1987

† Deptartment of Computer Science
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, New York 10012

‡ Computational Neuroscience Laboratories
Dept. of Psychiatry
NYU Medical Center
550 First Ave.
New York 10016 N.Y.

This research has been supported by the System Development Foundation, AFOSR #85-0341, and the Nathan S. Kline Research Institute.

"@(#)CIMS-TR-1-87 1.8 2/12/87 NYU Comp. Neuro Science Lab"

COMPUTING MINIMAL DISTANCES ON ARBITRARY POLYHEDRAL SURFACES

Estarose Wolfson
Eric L. Schwartz

Robotics Research
Courant Institute
Department of Computer Science
715 Broadway
New York, N.Y. 10003

Computational Neuroscience Laboratories
New York University School of Medicine
Department of Psychiatry
550 First Avenue
New York, N.Y. 10016

ABSTRACT

We have implemented an algorithm that makes iterative use of the law of cosines to find all the minimal (geodesic) distances in an arbitrary (that is, non-convex) three-dimensional polyhedral surface. The algorithm is intrinsically parallel, inasmuch as it deals with all nodes simultaneously. It has let us obtain very satisfactory flattening of biological (monkey visual cortex) surfaces consisting of several thousand triangular faces, by providing a full characterization of the distance geometry of these surfaces.

supported by System Development Foundation, AFOSR 85-0341 and the Nathan S. Kline Psychiatric Research Institute

COMPUTING MINIMAL DISTANCES ON ARBITRARY POLYHEDRAL SURFACES

Estarose Wolfson
Eric L. Schwartz

Robotics Research
Courant Institute
Department of Computer Science
715 Broadway
New York, N.Y. 10003

Computational Neuroscience Laboratories
New York University School of Medicine
Department of Psychiatry
550 First Avenue
New York, N.Y. 10016

Introduction

Several algorithms have been described recently that find minimal distances in polyhedral surfaces. An algorithm restricted to convex polyhedrons is described in (Sharir and Schorr, 1984), and an algorithm to find the shortest path between two points on an arbitrary polyhedral surface has been proposed in (Mitchell and Papadimitrious, 1984) and (O'Rourke et al, 1984).

We present a relatively simple algorithm for finding all the minimal distances between the nodes of an arbitrary polyhedron, which we have implemented, and tested on a fairly large scale problem posed by flattening the surface of the visual cortex of a monkey.

We developed this algorithm for this specific application: to characterize the metric structure of the cortical surface of the brain in primates and humans. We also

supported by System Development Foundation, AFOSR 85-0341 and the Nathan S. Kline Psychiatric Research Institute

want to "unfold" and flatten these brain surfaces. Apart from artificially created fractal surfaces, the folded, highly convoluted surface of the brain is one of the most complex surfaces encountered in computer graphics. Therefore, we had to develop an algorithm that was simple and practical, and that would not fail in the presence of highly complex data.

One of the reasons for computing the distances in such surfaces is to be able to "flatten" them. Elsewhere (Schwartz et al, 1987) we describe an algorithm that does this flattening by using a steepest-descent approach to minimize the difference between the "distance matrix" of the original surface and a planar model of that surface. Obviously, we cannot do any flattening until we obtain the original "distance matrix" : the set of inter-point distances implicitly characterizes the surface geometry. The algorithm we describe in this paper performs that task.

A certain synergy is evident in this work. In order to flatten a complex surface, we must first calculate many (if not all) of the distances in it. However, once we have obtained a quasi-isometric flattening, we can use this map itself to calculate distances. Because we can obtain good-quality flattening by calculating only a fraction of the entire set of distances, one possible strategy is to calculate a sparse-distance matrix (i.e., a matrix for the distances in local patches), use the distances in this matrix to flatten the polyhedral model, and then compute the rest of the (long range) distances from the planar model. If we use a sparse-distance matrix, we can efficiently represent the distances in it as a forest of binary-search trees.

Figure (0) shows how we applied a variational "flattening" algorithm to the visual cortex of a monkey. Figure (0a,b) shows a three-dimensional wire-frame model of the cortex, and Figure (0c) shows the "flattened" version of this data, according to the geodesic distances provided by the present algorithm. We computed the distances around each node at least as far as its tenth neighbor. This depth of search, although limited, nevertheless let us obtain excellent results from our flattening algorithm (Schwartz et

al, 1987).

We estimate the complexity of this algorithm as $O(N4^N)$, where N is the number of nodes in the polyhedral surface model. Fortunately, we can use the algorithm with a relatively limited neighborhood of distances (e.g., a 12-neighborhood) to obtain a planar model of a very complex surface. We can then use the planar model to calculate longer-range distances without incurring the computational expense of applying the algorithm to neighborhoods that lie farther from the given nodes. This procedure may have practical significance for other applications, such as robotics motion and autonomous vehicle navigation, that require the computation of geodesic distances.

Outline of the algorithm

We start with a polyhedron formed by convex planar polygons. Immediately we obtain first-neighbor distances: the given lengths of the edges of the polygons that form the polyhedron. We then apply the law of cosines to obtain the second-neighbor distances and any lengths and angles of the edges of the given polygons that we do not already know. To simplify the problem, we break the polygonal faces down into triangles. Any two neighboring triangles form a dihedral angle. To find the distance between any two nodes, we can rotate the triangles around the dihedral angle so that the two triangles lie in the same plane, and then apply the law of cosines.

However, we need not actually perform this rotation of the triangles about the dihedral angle. The given angles and edge lengths, together with the law of cosines, suffice to let us calculate the desired distances. Although it is helpful to think of the two triangles as lying in the same plane, it is not necessary to actually perform any computations to force them to do so. This fact lets us avoid a great deal of unnecessary computation.

At this point, we can specify a recursive use of the law of cosines, together with a growing list of angles and distances between nodes in the polyhederon. Some of

these angles and distances are the angles and edge lengths of the original polygon. Others, which are the result of computations performed during recursion, represent the angles and edges of new objects that we call "P-triangles" and "P-quadrilaterals." The "P" stands for "pseudo," because although all of the nodes of these figures do not really lie in the same plane (that is, dihedral angles exist between the triangles), for the purposes of this algorithm we can imagine that they do.

The first two steps in the recursive application of the law of cosines, as described above, are simple. However, as the level of recursion increases, we need to start considering special cases and possible complications. The following sections of this report define the new geometric objects that are essential to a discussion of the algorithm, and offer a proof of the algorithm's validity.

1. Definitions of the geometric objects used in the algorithm

1.1. Polyhedron

In R^3, a polyhedron is a figure defined to be a finite set of plane polygons such that every edge of a polygon is shared by exactly one other polygon (adjacent polygons) and no subset of polygons has the same property (Preparata and Shamos,1985). The nodes and edges of the polygons are the nodes and edges of the polyhedron. Figure 0 shows a polyhederal model of a brain surface.

In the following discussion, we will limit our consideration to polyhedrons that consist of triangular polygons. The term "triangle" refers to one of these polygons.

1.2. n-chain of triangles

An n-chain of triangles is an ordered list of triangles such that adjacent triangles in the list share a common edge, and such that the n-chain can be unrolled or "flattened" into a planar series of triangles with no overlapping edges in the plane. One major element of our algorithm is the construction of n-chains of triangles. In

figure 2a, triangles I, II, III, IV form a 4-chain. Figure 2c shows a 9-chain.

1.3. n-path of distances

An n-path between two nodes is an ordered list of edges along one or more n-chains, such that adjacent edges in the list share a common node.

1.4. P-triangle and P-quadrilateral

The following steps provide an inductive definition of these terms.

Step 1: A 1-chain

A 1-chain and a P-triangle is a triangle of the original polyhedral surface.

Step 2: 2-chains

A 2-chain of triangles (equivalently 2 P-triangles that share a common edge) form a P-quadrilateral (see Figure 1a), provided the quadrilateral formed by the two triangles is convex. (Figure 1b shows a non-convex example, which therefore is not a P-quadrilateral.) We find the unknown diagonal of the P-quadrilateral by applying the law of cosines. (We already know the other diagonal: it is the common edge of the P-triangles. We use the terms "distance" and "diagonal" interchangeably in the following discussion, because one of the sources of the distances in the computation is the set of diagonals of P-quadrilaterals.)

Note that this diagonal lies within the 2-chain of its P-quadrilateral (thus indicating that the P-quadrilateral is convex). We now define the P-triangles over the 2-chain to be all the P-triangles that can be formed by this new diagonal of the P-quadrilateral and the previously known edges of the P-triangles that lie within the 2-chain. In Figure 1a, both ADB and ACB are P-triangles over a 2-chain.

Step 3 (induction): So far, we have built all the possible P-triangles over n-chains. Now, given two chains m_1 and m_2, such that $m_1 + m_2 = n+1$, we form all possible P-quadrilaterals and find their unknown diagonals. These diagonals lie over (n+1)-chains. Now, using our previous set of diagonals over n-chains together with these

new diagonals, we can form P-triangles over (n+1)-chains (see Figure 2c).

1.5. Broken diagonals on n-chains

If the P-quadrilateral over a 2-chain is not convex, then one of its diagonals must lie outside the 2-chain. The distance between the nodes connected by this diagonal can be determined in either of two ways. One possibility is that it will be found later on another n-chain and P-quadrilateral. The other possibility is that the shortest distance between these two nodes is one of the 2-paths on the 2-chain† . In either case, we place this diagonal on a list of possible "broken diagonals." A later section of this report explains how the algorithm handles this list.

Then, as we build larger P-quadrilaterals, some of them will be non-convex. If two nodes of a P-quadrilateral are connected by a diagonal that lies outside the n-chain, then the shortest of the n-paths on the n-chain connecting these two nodes are placed on a list of possible "broken diagonals." Figure 3D shows examples of broken diagonals on longer paths.

2. The algorithm

2.1. Statement of algorithm

Given the preceding definitions, most of the algorithm is self-evident.

We iteratively build up the table of distances between nodes by building larger and larger P-triangles and P-quadrilaterals. The first iteration considers all 1-chains, and the second iteration considers all of the P-quadrilaterals over 2-chains. The third iteration considers all of the P-quadrilaterals over 3-chains, as well as the P-quadrilaterals over 4-chains which, depending on the data, can be built from pairs of 2-chains. At the n^{th} order of iteration, all P-quadrilaterals over n-chains are processed.

† Sharir and Schorr (1984) show that for a convex polyhedron, only the first of these two possibilities can occur.

as are some P-quadrilaterals over chains of order up to 2^{n-1}

The list of "broken diagonals" is maintained throughout the application of the algorithm. At any stage, however, only the shortest broken-diagonal length is kept on the list.

At a certain order of iteration, the algorithm stops and prunes the list of broken diagonals by removing from it the nodes that are represented by broken diagonals but that have yielded valid P-quadrilateral diagonals. The remaining broken-diagonal distances, which represent as the best available estimates of the distances between those nodes, are kept on the list.

Finally, the algorithm builds n-paths of diagonals that might consist partially of broken diagonals and partially of legitimate P-quadrilateral diagonals. Care is taken to consider all of the broken diagonals around each node.

Note that if distances between non-nodal points (i.e. which lie on the faces of triangles) are desired, simply add them to the data structure used to describe the polyhedron. This is trivial, since it amounts to triangulating a triangle with an added internal point.

2.2. Complexity

We can build an (n+1)-chain in only four ways. This is because an n-chain (n >1) by definition has only four "free" edges to which we can attach a new figure. Therefore, because the algorithm terminates at a point that does not exceed O(N) (The maximal length of any n-chain is O(N), where N is the number of nodes of the polyhedral surface), and because the polyhedral surface contains O(N) triangles, the complexity is O ($N4^N$) .

2.3. Verification of the algorithm

Given the preceding definitions and discussions, our geodesic-distance-determining algorithm is trivially correct. Nevertheless, the following proof may be of

interest.

Theorem: Given a P-quadrilateral over an n-chain, there exists a diagonal that lies within the said n-chain.

Conversely, given an n-chain, if there exists a straight line (i.e., a line drawn on the flat model of the n-chain), that lies within the n-chain and thus crosses all the triangles making up the n-chain, then there exists a P-quadrilateral whose diagonal is this line.

Proof:

=> By definition of P-quadrilateral convexity.

<= Choose the closest node of the n-chain on either side of the given straight line. In Figure 4a, these are nodes C and D. Thus we have found two triangles, whose common edge is line CD and those third nodes are the end-points of the given straight line. The union of these two triangles is a quadrilateral.

The angles of the quadrilateral in Figure 4a are less than 180 degrees, because the given diagonal lies within the quadrilateral as constructed.

Both triangles lie within the n-chain. If they did not, then the assumption that their vertices are the closest points to the given straight line would be contradicted (see Figure 4b). Thus, we have found the required P-triangles and P-quadrilateral.

2.4. Discussion of the algorithm

The foregoing theorem proves that if a straight-line path exists between two nodes, exhaustive construction of P-triangles and P-quadrilaterals will reveal it. Any distance that is not found this way must be a "broken diagonal." If the algorithm is iterated O(N) times, then all of the shortest distances will necessarily be found.

In practice, we run the algorithm far short of all O(N) possible iterations (N typically being on the order of 1000). We only need to calculate distances on "patches" containing about 12 nodes in order to obtain successful flattening from random starting

configurations of our flattening algorithm (Schwartz et al, 1987). For limited runs such as these (which, given the complexity of the algorithm, are in fact the only feasible runs), the best we can expect is a good approximation to the set of minimal distances in the neighborhood. It is possible that a pathological surface (e.g., one with many bumps and valleys of just the right size) could have a minimal path of, say, neighborhood size 11, which would be slightly shorter than the path we actually find (through the broken-distance list) for a neighborhood size of 12. In practice, however, this limitation has not been a problem. Our flattened-distance estimates usually match our three-dimensional polyhedral edge-lengths to within a few percent† .

In this sense, we use this algorithm as an heuristic approximation to a minimal geodesic-distance algorithm. In our application, running this algorithm iteratively up to a limit of small N does seem to yield very good approximations to the geodesic distances that we need in order to obtain accurate flattening of the cortical surfaces of the brain.

It is worth pointing out that we have implemented two versions of this algorithm. The first version is the one described above, where at the m^{th} iteration, n-chains up to length 2^{m-1} can possibly be obtained. The other version, at the m-th iteration, is limited to n-chains of length m. We have used both runs and merged them for later flattening, thus obtaining a sampling of very long-range distances and also spanning a large neighborhood around each node. Limitations of machine time and storage space make this procedure worthwhile.

3. Performance

For a polyhedron consisting of about 2500 triangles representing the surface of monkey striate cortex, we calculated all of the distances over 12-chains on each node.

† Obviously, flattening a closed surface, or one which is "close" to being closed, will not provide a good quasi-isometry. In practice, our surfaces correspond to small fractions of the solid angle of a sphere, and therefore can, in principle, yield good quasi-isometries.

On a Sun-2 workstation with 4 megabytes of physical memory, this run took about 6 hours and used 12 megabytes of virtual memory. The run-time would have been much shorter if a full 12 megabytes of real memory had been available. The use of virtual memory rather than real memory meant that most of the run-time was given over to thrashing. Nevertheless, this run (in conjunction with the use of other software) resulted in the successful flattening of the surface of the striate cortex (Schwartz et al, 1987)

4. Other Applications

This algorithm finds local shortest-distance patches. Used together with our flattening algorithm, it lets us measure overall shortest distances. This may in fact be the most efficient way to compute long-range shortest distances on the class of polyhedra whose global curvature allows flattening with a relatively small error† .

† In other work we describe ways to measure the mean and Gaussian curvature of polyhedra (Kaplow and Schwartz, 1986).

Figure Captions

Figure 0. Two different views of a polyhedral model of primate visual cortex, consisting of about 2500 trianglular faces. These wire frame models are very complex and difficult to view; an easier view of the structure is presented in figure 0.C, which is the polyhedron unfolded and flattened, by an algorithm described in other work(Schwartz et al, 1987). This flattening is the purpose for the geodesic distance calculations of this report.

Figure 1. P-quadrilateral ACBD formed from P-triangles ACD & BCD with common edge CD. The P-triangles are opened along edge CD so that they lie in the same plane.

In Figure 1a, the angles ACB & ADB are both < 180 degrees. Therefore, diagonal AB lies within the P-quadrilateral, and a new diagonal or distance is found between nodes A & B. We find all of the sides of the P-triangles in the first iteration of the algorithm. We find the angles by applying the law of cosines. We then find the new diagonal by applying the law of cosines again, using AC & CB and angle theta1+theta2.

In Figure 1b, angle ADB is > 180 degrees. Therefore, diagonal AB lies outside the n-chain. Thus, the quadrilateral is not a P-quadrilateral, and no distance is found here.

Figure 2. P-quadrilateral ACBD over 9-chain I, II, III, IV, V, VI, VII, VIII, IX (Figure 2c) is formed from P-triangle ADC with sides AD over 3-chain I, II, III and AC over 4-chain I, II, III, IV (Figure 2a) and P-triangle CDB with sides BC over chain IX, VIII, VII, VI and BC over chain IX, VIII, VII, VI, V. This P-quadrilateral generates diagonal AB.

Figure 3. "Broken distances" between nodes. Figure 3a is a three-dimensional view of a node P where the sum of the angles is > 360 degrees. Thus, certain flattenings of the polygons around this node will produce angles that are > 180 degrees in both directions. Therefore, no new diagonals can be found at this level, and the

distance is a "broken diagonal": APE in Figure 3b and Figure 3c. All the different possible "broken distances" must be examined in order to determine the shortest distance between nodes (for example, ACB, ADB, AEB, AFB, AGHB for distances between nodes A & B, as shown in Figure 3d).

Figure 4. Given the diagonal AB within an n-chain, we can find a P-quadrilateral that meets our specifications and generates the diagonal. Dropping perpendiculars to the diagonal AB from each of the other nodes C, D, E, F, G, H, I, J, we find node C is closest to AB on one side and node C is closest to AB on the other side. Thus, P-quadrilateral ACBD is formed from the P-triangles ADC and BCD having the common edge DC (Figure 4a). In the proof, if edge AC did not lie on the n-chain of diagonal AB, there would be another node (say, E or F) that is nearer to AB and on the same side as node C.

References

Mitchell, S.S.B. and C.H. Papadimitrious, "The discrete geodesic problem," *Stanford University Technical Reports*, 1984.

O'Rourke, J., S. Suri, and H. Booth, "Shortest paths on polyhedral surfaces," *Tech. Report*, Dept. E. Engineering, Johns Hopkins Univ., 1984.

Schwartz, E.L., A. Shaw, and E. Wolfson, "A numerical solution to the generalized map-maker's problem: Flattening non-convex polyhedral surfaces.," *Courant Institute Tech. Reports*, vol. CIMS-TR-#256, 1987.

Sharir, M. and A. Schorr, "On shortest paths in polyhedral surfaces," *Siam J. Comput.*, vol. 15 No. 1, pp. 193 -215, Society for Indust. and Applied Math., New York, NY, 1984.

Preparata, Franco P. and Michael Ian Shamos, *Computational Geometry: An Introduction*, Springer Verlag, New York, 1985.

Kaplow, W.K. and E.L. Schwartz, "Measuring mean and Gaussian curvature on triangulated brain surfaces: The differential geometry of macaque striate cortex." *Comp.Neuro.Tech.Rep.*, vol. CNS-TR-1-86, 1986.

FIGURE 0a

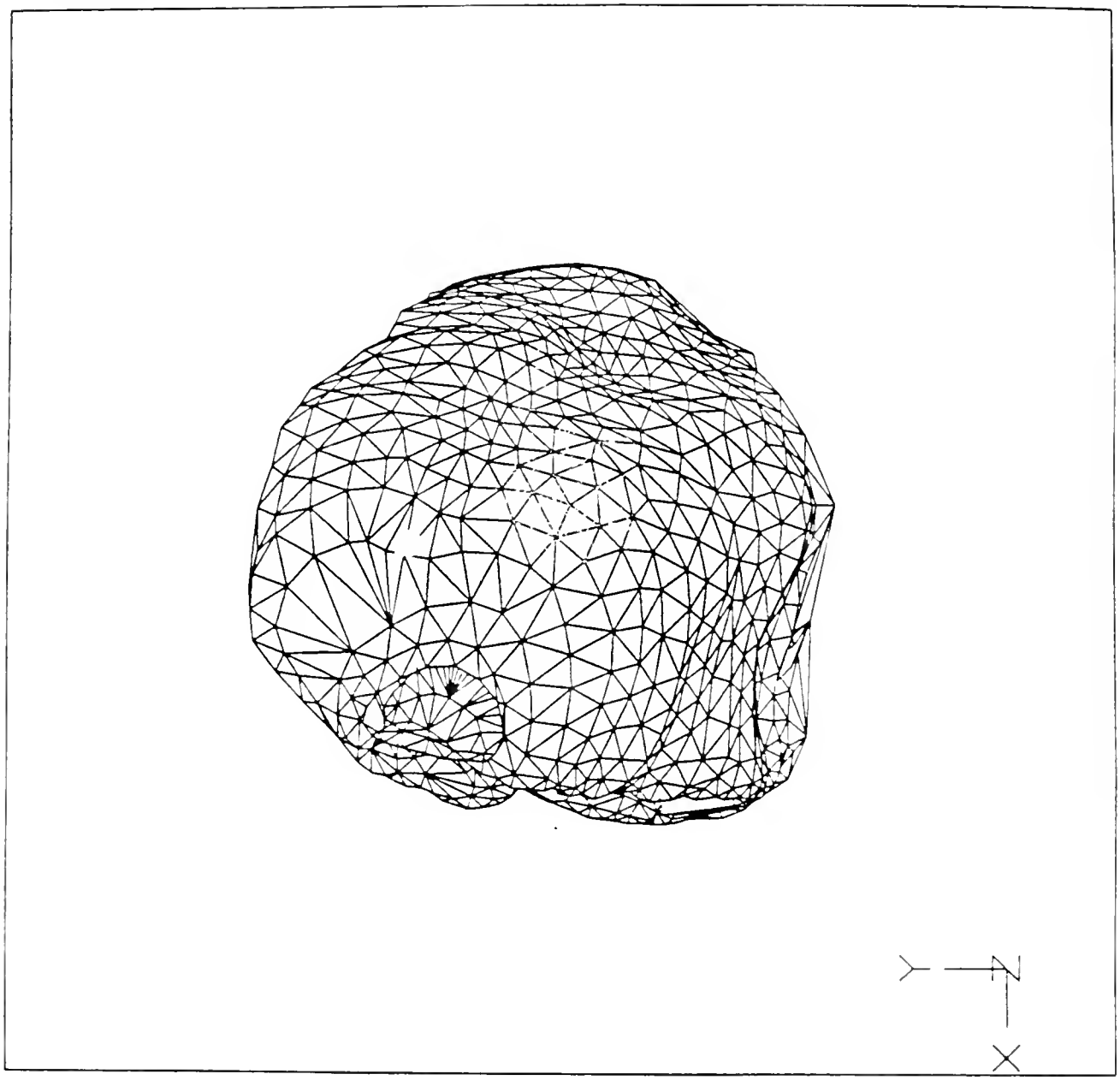


FIGURE 0b

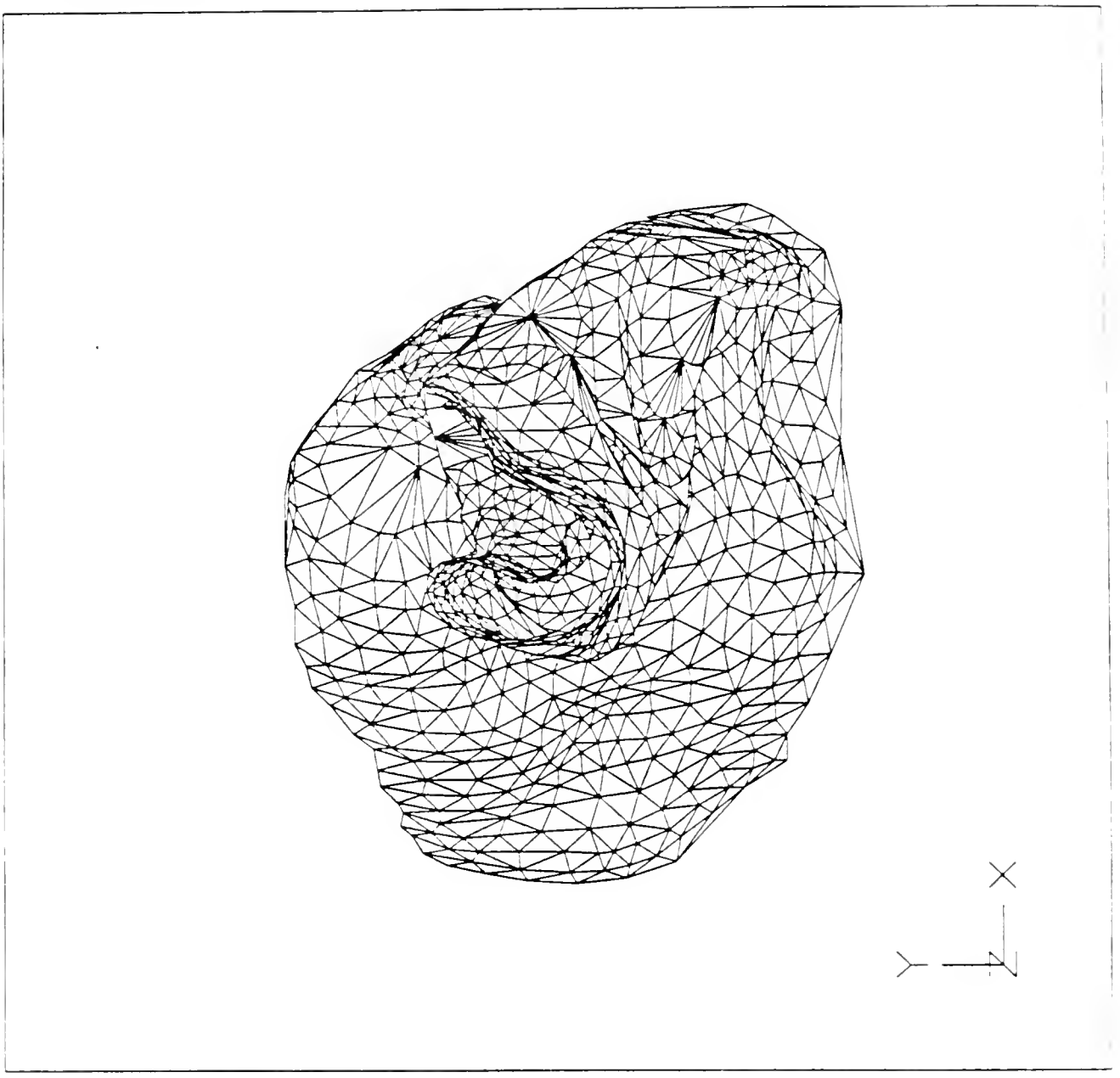


FIGURE 0c

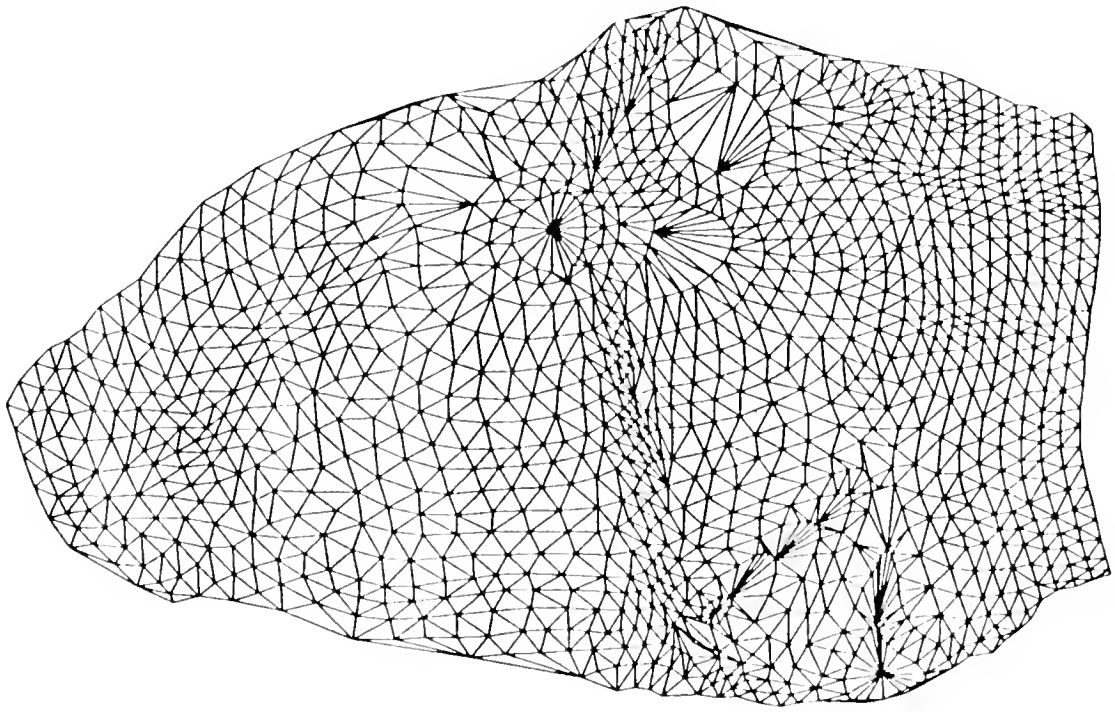


FIGURE 1

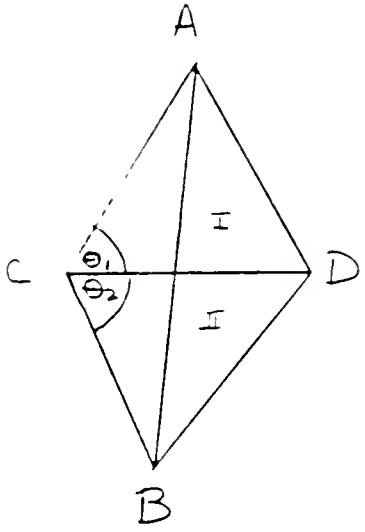


Figure 1a

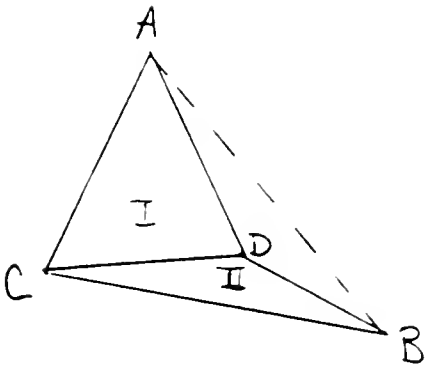


Figure 1b

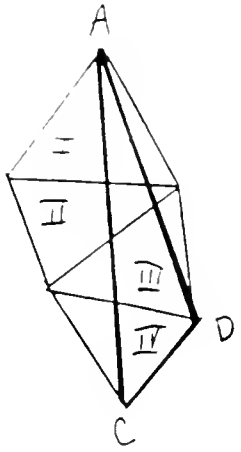


Figure 2a

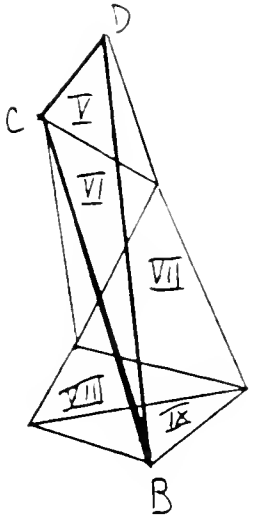


Figure 2b

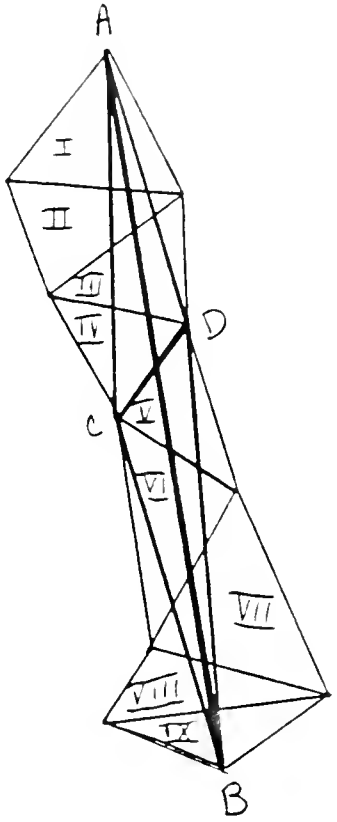


Figure 2c

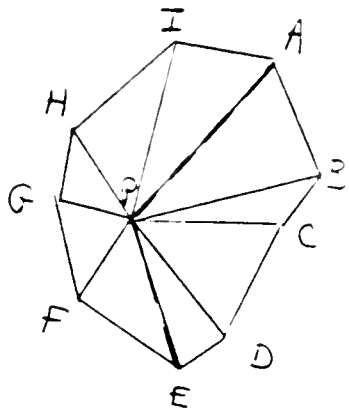


Figure 3a

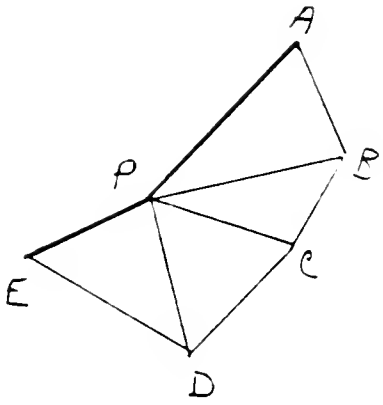


Figure 3B

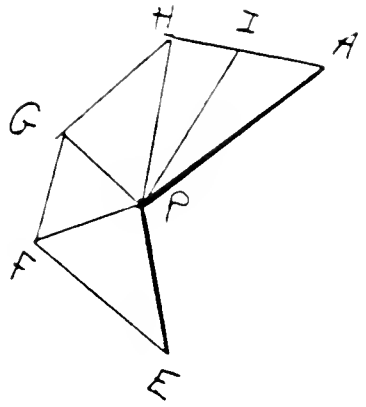


Figure 3C

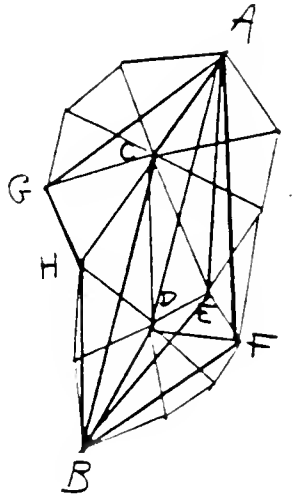


Figure 3D

FIGURE 4

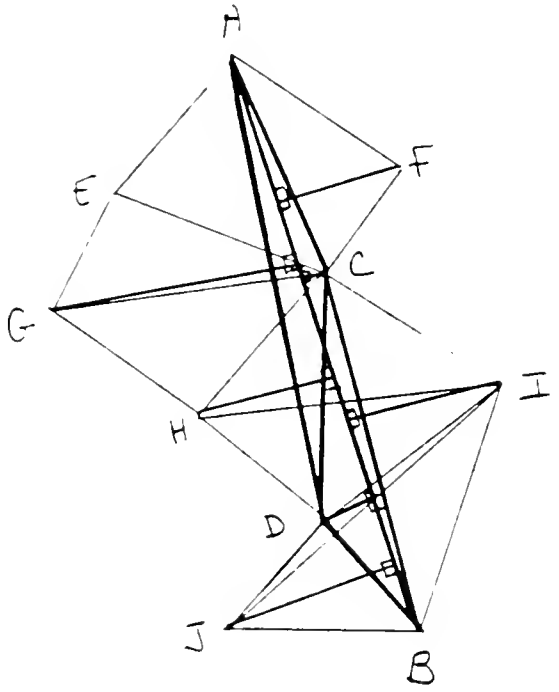


Figure 4a

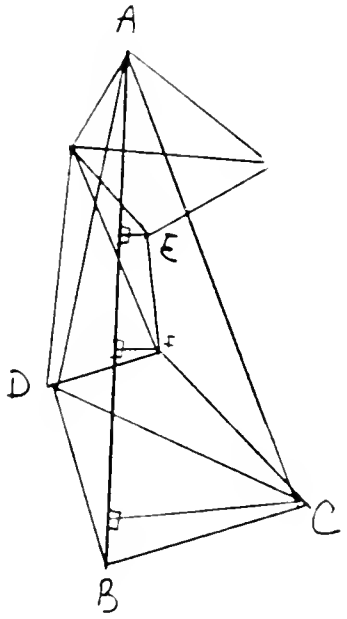


Figure 4b

NYU COMPSCI TR-274 c.1
Wolfson, Estarose
Computing minimal distances
on arbitrary polyhedral
surfaces.